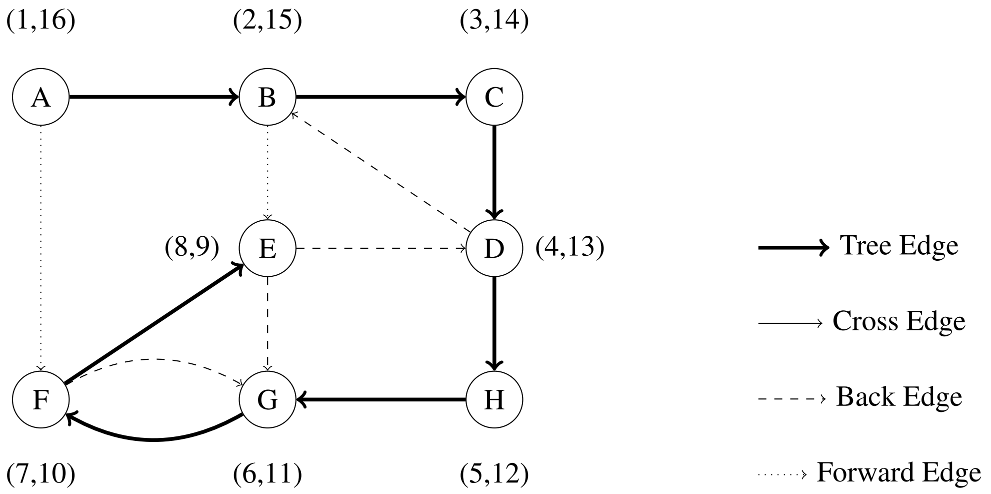
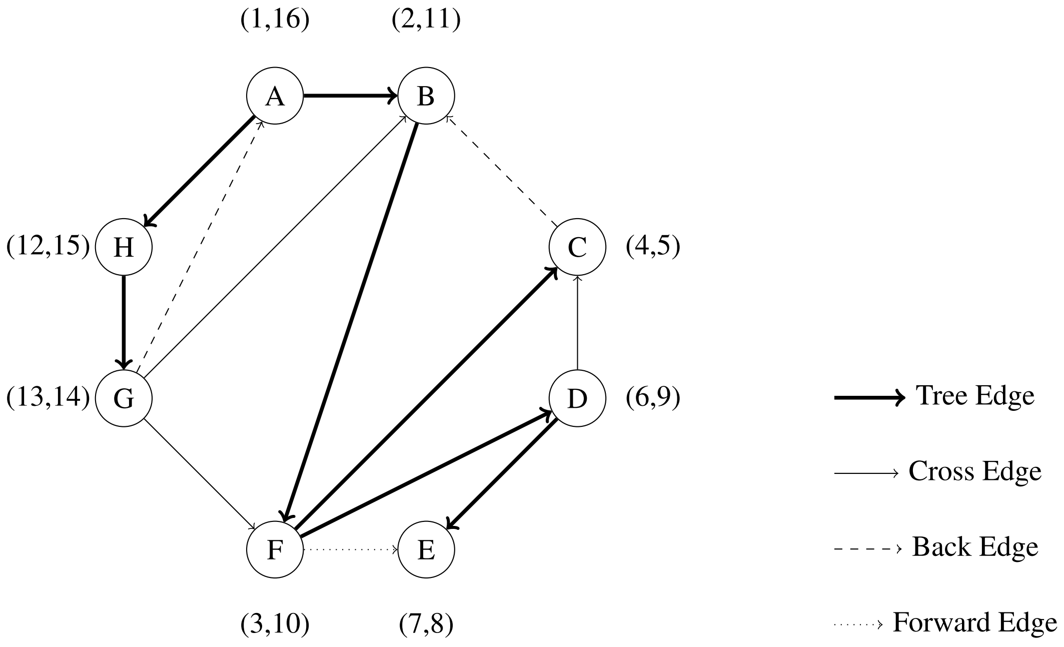
CMPSC 465 Fall 2023 | Data Structures & Algorithms Antonio Blanca and Yana Safonova | HW 5

1. (21 pts.) Problem 1

(a)



(b)



2. (21 pts.) Problem 2

(a) $A: 1, 14$
$B: 15, 16$
$C: 2, 13$
$D: 3, 10$
$E: 11, 12$
$F: 4, 9$
$G: 5, 6$
$H: 7, 8$

(b) Sources: A, B; Sinks: G, H

(c) B, A, C, E, D, F, H, G

(d) Any ordering of the graph must be of the form $\{A,B\}, C, \{D,E\}, F, \{G,H\}$, where $\{A,B\}$ indicates A and B may be in any order within these two places. So, the total number of orderings is $2^3 = 8$.

3. (20 pts.) Problem 3

(a) We create a graph $G = (V, E)$ where vertices represent locations and edges represent roads. Because the roads are two-way, the graph must be undirected. In this graph, locations that can be driven between are represented by connected components. The graph might have several connected components, separated by the "rivers." An edge (u, v) in this graph is safe to remove if it does not separate its connected component. Because the edge connects u to v, u and v must start in the same connected component. If (u, v) is removed and v is still reachable from u, then the connected component must have contained a cycle that included (u, v). This logic is the reverse of worksheet 5 problem 4. So there is some road that is safe to close if and only if our graph G has a cycle.

(b) We can't just use DFS to check for cycles, since that takes time $O(|V| + |E|)$. However, we can modify DFS to terminate the first time we see an edge that goes back to a visited vertex; this is a minor modification to the Explore procedure. This algorithm will detect if there is a cycle.

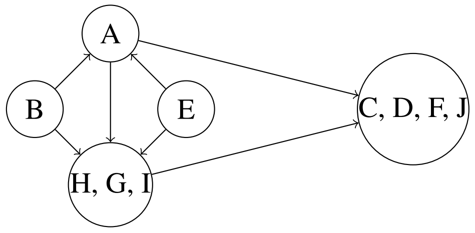
Let us analyze the running time of this modified DFS algorithm. The key observation is that a graph with more than $|V|$ edges will always have a cycle. (If you don't see why, you should prove this fact. Essentially, if $|E| \geq |V|$ you are forced to create a cycle.) Suppose first that the graph has no cycles. Then, $|E| < |V| - 1$. So, in this case, the running time of $O(|E| + |V|)$ of DFS is actually $O(|V|)$. Now, suppose that the algorithm stopped early. This is because it found some edge coming from the currently considered vertex to a vertex that has already been considered. Since all of the edges considered up to this point didn't do that, we know that they formed a forest. So, the number of edges considered is at most the number of vertices considered, which is $O(|V|)$. Consequently, the total running time is $O(|V|)$.

4. (20 pts.) Problem 4

(i) (a) The strongly connected components are found in the order $\{C, D, F, J\}$, $\{G, H, I\}$, $\{A\}$, $\{E\}$, $\{B\}$.

(b) The source SCC's are $\{E\}$ and $\{B\}$, while $\{C, D, F, J\}$ is a sink SCC.

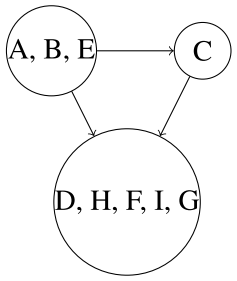
(c)



(d) It is necessary to add two edges to make the graph strongly connected, e.g. by adding $C \to B$ and $B \to E$.

(ii) (a) The strongly connected components are found in the order $\{D,F,G,H,I\}$, $\{C\}$, $\{A,B,E\}$.

(b) $\{A,B,E\}$ is a source SCC, while $\{D,F,G,H,I\}$ is a sink.

(c) 

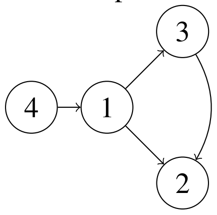
(d) In this case, adding one edge from any vertex in the sink SCC to any vertex in the source SCC makes the metagraph strongly connected and hence the given graph also becomes strongly connected.

5. (19 pts.) Problem 5 **Solution**

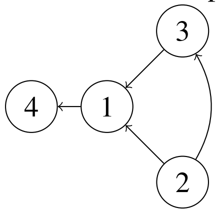
(a) We construct the directed graph $G = (V,E)$, where every node represents a student (numbered 1 to n), and every edge (u,v) represents u being friends with v. If v is also friends with u, another edge (v,u) will also be present. In this graph, the first-pick partner of student s is the node of minimum value reachable from node s.

(b) Here, if we reverse the direction of every edge in the graph, the set of students who have good student k in their social network (k is reachable), becomes precisely the set of students that good student k can reach. Furthermore, all vertices within the same SCC should ultimately receive the same label (the best student in that SCC), along with all the vertices within any other SCC that our current SCC can reach. Thus, for all unvisited nodes in the reverse graph, we can run DFS starting from the lowest-ranked unvisited node i, and for all the nodes j that are reachable from i, we assign i to be student j's first-pick partner.

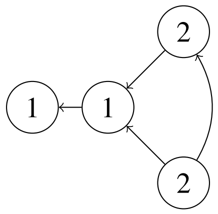
For example, if we start with the graph:



We can see that upon reversing it we'd get the graph:



When we run DFS on this graph, we'll need two 'starts'. Once starting at 1 (the best-ranked student) and then again starting at 2 (the best-ranked student we haven't visited yet) after we hit a dead end. Our two runs produce the following labels:



Looking back at our original graph, we can see that this does indeed correspond to the right answer!

We can see that this algorithm is very similar to our algorithm for finding SCCs. Rather than starting from a vertex in a sink SCC, however, we start from the vertex with the smallest value. So rather than finding SCCs one at a time, we'll first find all the vertices in the SCCs that can reach 1, then all the vertices in SCCs that can reach the next smallest available number, and so on.

Algorithm

Let G^R be the graph G with its edge directions reversed. The algorithm is as follows.

```
Function Choosing-Partners($G$):
    while there are unvisited nodes in $G^R$ do
        Run DFS on $G^R$ starting from the numerically-first unvisited node $i$
        for $j$ visited by this DFS do
            first_pick[$j$] := $i$
        end
    end
```

To see that this algorithm is correct, note that if a vertex i is assigned a value then that value is the smallest of the nodes that can reach it in G^R, and every node is assigned a value because the loop does not terminate until this happens. Now observe that the set of vertices reachable by i in G^R is the set of vertices which can reach i in G.

The running time is $O(|V|+|E|)$ since computing G^R can be done in linear time, and we process every vertex and edge exactly once in the DFS.

Rubric:

Problem 1, 21 pts

(a)
- 0.5 pts for every node's pre number and post number
- 0.5 pts for every edge type

(b)
- 0.5 pts for every node's pre number and post number
- 0.5 pts for every edge type

Problem 2, 20 pts

(a) 8 points: 0.5 point for a correct pre number and a correct post number, respectively, for each vertex.

(b) 2 points: 0.5 point for each correctly categorized vertex.

(c) 4 points: 0.5 point for each correctly ordered vertex.

(d) 6 points
1 point: find a pair of nodes that can be swapped in other correct topological ordering
2 points: provide the correct final answer.
1 point: provide an explanation on how those pairs of nodes lead to the correct answer.

Problem 3, 20 pts

(a)
- 2 pts: Undirected graph where nodes represent locations and edges represent roads.
- 3 pts: States that "safe-to-close roads" implies finding cycles.

(b)
- 5 pts: Relating the back edge found by DFS to existence of a cycle in undirected graph.
- 5 pts: modifying the Explore procedure such that it finds a back edge, and terminates early (just an explanation would be enough, they do not need to write a pseudocode)
- 5 pts: analyzing the runtime of the algorithm, and explain why it would be independent of $|E|$ if we terminate the algorithm early

Problem 4, 20 pts

(a) 2 points for (i) and (ii), respectively

(b) 2 points for (i) and (ii), respectively

(c) 4 points for (i) and (ii), respectively

(d) 2 points for (i) and (ii), respectively

Problem 5, 19 pts

(a)
- 2 pts for directed graph
- 2 pts for the right description of node
- 2 pts for the right description of edge
- 3 pts for the questions raised by this problem

(b)
- 5 pts for reverse graph and some indication of why it's helpful
- 5 pts for exploring the reverse graph from the next best-ranked student (node), marking all reachable nodes with that student's number.